Determine whether Family Tree relation exists or not

A family tree is a chart representing family relationships in a conventional hierarchical tree structure. Family trees are often presented with the oldest generations at the top and the newer generations at the bottom.

Family tree can have many relations. Given the family tree and a relation among two family members you need to write a program to print whether the relation is true or false.

**Definitions:**

1. **Parent:** Person A is parent of B, if A and B are directly connected and B is exactly one level below to A in the family tree. E.g. In Fig 1. Motilal is parent of Jawahar, Jawahar is parent of Indira, etc.
2. **Sibling:** Siblings are members that have the same immediate parent. E.g. In Fig.1 Rahul & Priyanka are Siblings, Sanjay & Rajeev are siblings, etc
3. **Descendant:** Person A is a descendant of Person B, if A is connected to B either directly or through any other person but is atleast one level below B in the family tree. E.g. In Fig 1. The descendants of Jawahar are Indira, Sanjay, Rajeev, Varun, Rahul & Priyanka. Kamala does not have any descendants.
4. **Ancestor:** Person A is a ancestor of Person B, if A is connected to B either directly or through any other person but is atleast one level above B in the family tree. E.g in Fig 1. Indira is ancestor of Sanjay, Rajeev, Varun, Rahul & Priyanka

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

typedef struct Node {

char\* value;

struct Node\* left;

struct Node\* right;

}Node;

Node \*root = NULL;

Node\* notnull ( Node \*node1, Node\* node2 ){

if( node1 != NULL)

return node1;

else if (node2 != NULL)

return node2;

return NULL;

}

Node\* search( char \*p, Node\* root){

if( root == NULL )

return NULL;

else if ( strcmp (root->value,p) == 0){

return root;

}

return notnull ( search(p,root->left), search (p, root->right) );

}

int insert(char\* parent,char\* child){

if( root == NULL ){

root = malloc ( sizeof( Node ) );

root->left = NULL;

root->right = NULL;

root->value = (char\*) malloc ( strlen(parent) + 1);

strcpy ( root->value, parent);

root->left = malloc ( sizeof( Node ) );

root->left->value = (char\*) malloc ( strlen( child ) + 1);

strcpy ( root->left->value, child);

return 1;

}else{

Node\* p = search( parent, root);

if( p == NULL){

return 0;

}

else if ( p->left == NULL ){

p->left = malloc ( sizeof( struct Node ) );

p->left->left = NULL;

p->left->right = NULL;

p->left->value = ( char \* ) malloc ( strlen ( child ) + 1 );

strcpy ( p->left->value, child );

return 1;

}else if ( p->right == NULL && strcmp(p->left->value,child) != 0 ){

p->right = malloc ( sizeof( struct Node ) );

p->right->left = NULL;

p->right->right = NULL;

p->right->value = ( char \* ) malloc ( strlen( child ) + 1 );

strcpy ( p->right->value, child );

return 1;

}

}

return 0;

}

int notzero(int x,int y){

if( x != 0 || y != 0 )

return 1;

return 0;

}

int isSibling(char \*name1,char \*name2, Node \*root){

if( root == NULL)

return 0;

else{

if ( root->left != NULL && strcmp (root->left->value,name1) == 0

&& root->right != NULL && strcmp(root->right->value,name2) == 0 ){

return 1;

}else if ( root->left != NULL && strcmp (root->left->value,name2) == 0

&& root->right != NULL && strcmp(root->right->value,name1) == 0 ){

return 1;

}

}

return notzero ( isSibling( name1,name2,root->left), isSibling( name1,name2,root->right) );

}

void printTree(Node \*root){

if( root != NULL){

printf("%s ",root->value);

printTree( root->left );

printTree( root->right );

}

}

void preorder(){

printTree(root);

printf("\n");

}

int main(int arn, char\*\* args){

int n;

scanf("%d\n",&n);

char record[n][2][26];

int i=0;

int flag = 0;

for( i = 0; i < n; i++){

scanf ( "%s %s\n",record[i][0],record[i][1]);

flag = 1;

}

while ( flag ){

for ( i = 0;i < n; i++ ){

if ( insert ( record[i][0], record[i][1] ) == 1 ){

i = 0;

continue;

}

}

flag = 0;

}

scanf("%d\n",&n);

for( i=0; i<n; i++ ){

char name1[26];

char name2[26];

char rel[15];

scanf ("%s %s %s\n",name1,rel,name2) ;

//printf("%s %s %s\n",name1,rel,name2);

if ( strcmp ( rel,"child") == 0 ){

Node \*p = search ( name2, root );

if( p != NULL ){

if ( p->left!=NULL && strcmp ( p->left->value,name1) == 0 )

printf("T ");

else if ( p->right != NULL && strcmp ( p->right->value,name1) == 0 )

printf("T ");

else

printf("F ");

}else{

printf("F ");

}

}else if ( strcmp ( rel,"parent" ) == 0 ) {

Node \*p = search ( name1, root);

if( p != NULL ){

if ( p->left != NULL && strcmp ( p->left->value,name2) == 0 )

printf("T ");

else if ( p->right != NULL && strcmp ( p->right->value,name2) == 0)

printf("T ");

else

printf("F ");

}else{

printf("F ");

}

}else if ( strcmp ( rel,"sibling" ) == 0 ) {

//printf("%s %s %s\n",name1,rel,name2);

if( isSibling ( name1, name2, root) == 1)

printf ("T ");

else

printf ("F ");

}else if ( strcmp ( rel,"descendant" ) == 0 ){

Node \*p = search ( name2, root);

if ( p != NULL ){

Node \*d = search ( name1, p);

if( d != NULL ){

printf("T ");

} else{

printf("F ");

}

}

else{

printf("F ");

}

}else if ( strcmp ( rel,"ancestor" ) == 0 ){

Node \*p = search ( name1, root);

if ( p != NULL ){

Node \*d = search ( name2, p);

if( d != NULL ){

printf("T ");

} else{

printf("F ");

}

}

else{

printf("F ");

}

}

}

printf("\n");

preorder();

return 0;

}